

F-SERIES Flatbed SDK

Revision History

Dec 2010: Original Issue.
Feb 2011: Added Foil Cut.
Feb 2011: Added Wii Remote.
May 2012: Added tools
June 2012: Removed Foil Cut added Z position (updated subset according to supported commands in firmware revision 008 – current version)
March 2015 removed DMPL support.
June 2018 added two new commands for miller removed detailed explanation of OXF format.

Notice

Summa reserves the right to modify the information contained in this document at any time without prior notice. Unauthorized copying, modification, distribution or display is prohibited. All rights reserved. Please address all questions, comments or suggestions concerning this manual.

Summa
Rochesterlaan 6
B-8470 Gistel
Belgium

Copyright © Summa
All names and trademarks are the property of their owners.

Table of Contents

1	Introduction	2
2	Writing a Native driver.....	4
2.1	HP-GL Language Basics.....	5
2.1.1	IN; Initialize Command.....	5
2.1.2	BP; Begin Plot Command (only HP-GL/2)	5
2.1.3	PA; Absolute Addressing Command	5
2.1.4	PR; Relative Addressing Command	5
2.1.5	PD x,y; Vector Down Command.....	5
2.1.6	PU x,y; Vector Up Command	5
2.1.7	SP tool, [slot_number]; Tool Select Command.....	6
2.1.8	SQ tool; Tool check Command	7
2.1.9	VS n; Velocity Command	7
2.1.10	RP n; router rotation speed Command	7
2.1.11	BC; Bit change command Command	7
2.1.12	ZP n,m; Z axis position Command	7
2.1.13	OH; Output Hardclip Command.....	7
2.1.14	OA; Output Actual Position	7
2.1.15	FF; Feed Command and FLx; command	8
2.1.16	PB Vacuum control	9
2.1.17	PS Vacuum direction.....	9
2.1.18	ZT0; Park heads	9
2.1.19	AUbeta; define lifting angle	9
2.1.20	OVn; Overcut distance.....	9
2.1.21	LMn; Local move commands.....	9
2.1.22	MStext MESSAGE.....	10
2.1.23	Sample HP-GL File.....	10
2.2	Encapsulated Language	11
2.3	Contour Cutting.....	11
2.3.1	Mark recognition using Custom software.	11
2.3.2	Mark recognition using SummaFlex Pro.	11
2.4	USB	12
2.4.1	Introduction.....	12
2.4.2	USB and Windows.....	12
2.4.3	USB Pipes.....	12
2.4.4	Win32 Software Components	13
2.4.5	SummaUsb.sys	14
2.4.6	SummaUsb.dll	14
2.4.7	Win32 Host Application.....	15
2.4.8	Plug And Play Considerations	15
2.4.9	Summary	16
2.4.10	Win32 Sample Application	16
2.4.11	Handshake Sample.....	21
2.5	Wii Remote	24
3	The OXF format	25

1 Introduction

The information contained in this package is designed to allow technically competent people to develop software and drivers compatible with the F series flatbed cutters.

There are 2 options to drive the machine:

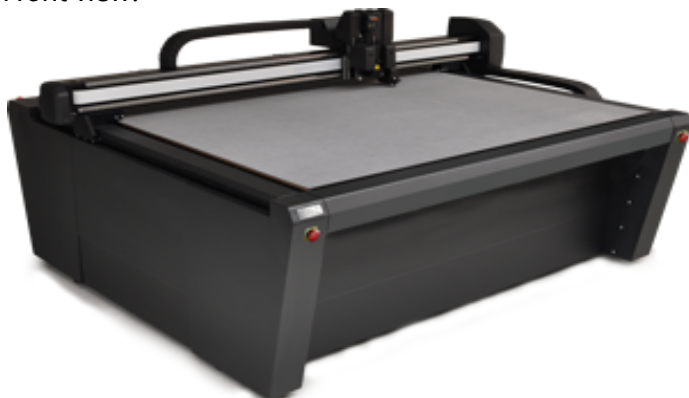
1. Write a native driver which communicates directly to the machine via the USB port. If contour cutting is needed, then mark recognition and compensation for contour cutting must also be written, a USB camera supporting MS DirectShow is installed on the machine, this camera is for the moment only compatible with Microsoft Windows.
2. Write a driver that creates files which can be imported by SummaFlex Pro. This is a commercial digital finishing application sold by Summa, it is a special version of OptiSCOUT. This version supports the OXF format written by Eurosystems. It also supports other formats such as pdf, eps, ai, dxf... SummaFlex Pro supports mark recognition and compensation for contour cutting as well as other advanced workflow features. The OXF format is described at the end of this document.

The F series flatbed cutter has following features:

- 3 slots on the cutting head where different kind of tools can be mounted. The tools are for example, cut out knives, kiss cutting knives, creasing wheels, router, oscillating knife, pen. These tools can be used for various media: flexible roll media or rigid materials.
- A camera used for mark recognition. If contour cutting is required, then some registration marks must be printed along the design. The application can then detect those markers and adapt the cutting path accordingly. The firmware of the F series does not support the well-known OPOS technology. It does however support OptiSCOUT by means of the program SummaFlex Pro.
- Optional conveyor and media clamps for feeding material through the machine.
- A vacuum pump that creates a vacuum so that the media "sticks" to the cutting surface. This vacuum can be reserved to blow the media slightly. This for easier media handling.
- A roll support to place a roll of media.

Some pictures:

Front view:



Rear view



Cutting head with camera/laser pointer/cutting tools.



2 Writing a Native driver

The cutter understands HP-GL and HP-GL/2. Because HP-GL and HP-GL/2 are almost identical, no difference is made in this document between both languages. Only a small subset of these languages is described. Additions have been made to these languages to support all the features of the flatbed cutter. It is strongly advised to use HPGL.

The **X-axis** is the direction in which the **top beam moves. In which the media is fed through the cutter.** The maximum size of media in X-direction is 50 meter, when feeding media is possible; otherwise it is the length of the table, for the F1612, this is 1200 mm.

The **Y-axis** is the direction in which the **cut-head** moves, the maximum size for the F1612 is 1600 mm.

The **origin** is at the **front right**, the front right is where the on/off switch is located.

2.1 HP-GL Language Basics

Only a subset of HP-GL and HP-GL/2 is presented in this document. "Arc/curve" commands are not supported.

The **standard resolution** for most HP-GL commands is 0.025 mm, however for the F series it is **0.01 mm**.

Note: All commands must end with a semicolon;

2.1.1 IN; Initialize Command

The *IN*; initialize command resets all parameters that were changed by HP-GL commands.

2.1.2 BP; Begin Plot Command (only HP-GL/2)

The *BP*; Begin Plot command resets all parameters that were changed by HP-GL/2 commands.

2.1.3 PA; Absolute Addressing Command

The absolute addressing *PA*; command selects the Absolute addressing mode. In this mode, all x-, y-co-ordinates are with respect to the origin at the lower left corner of the chart. The device remains in this mode until it receives a Relative *PR* command.

Example: *IN*; *PA*; *PU* 5000,5000;*PD* 2000,2000;

In this example, the cutter is initialized with the *IN*; command. Absolute knife positioning is selected with the *PA*; command. The knife is sent in "up position" to absolute co-ordinate 5000,5000. The knife is lowered with the down *PD* command and a line is cut to absolute co-ordinate 2000,2000.

2.1.4 PR; Relative Addressing Command

The relative *PR*; command selects the Relative addressing mode. In this mode, all x-, y-co-ordinates are relative to the present knife position. The device remains in this mode until it receives an Absolute *PA* command.

2.1.5 PD x,y; Vector Down Command

The down *PD*; command lowers the tool. This command causes the knife to move to a new user position specified by co-ordinate x,y. This new position is dependent on the present Absolute *PA* or Relative *PR* knife positioning command in effect. In Absolute *PA* mode, the new position is x,y user units from the present origin. In Relative *PR* mode, the new position is x,y user units from the present position.

2.1.6 PU x,y; Vector Up Command

The Up *PU*; command raises the knife. This command causes the knife to move to a new user position specified by co-ordinate x,y. This new position is dependent on the present Absolute *PA* or Relative *PR* knife positioning command in effect. In Absolute *PA* mode, the new position is x,y user units from the present origin. In Relative *PR* mode, the new position is x,y user units from the present position.

2.1.7 SP tool, [slot_number]; Tool Select Command

The *SP tool;* command selects the tool to be used. (e.g. knife, pen, router, creasing wheel...). The machine has 3 positions, called slots, where the tool can be mounted. The tools are automatically detected and selected when receiving the SP command. If a tool is not available on the Flatbed the job will be aborted. So it is advised to first use the SQ command to prevent this error. The optional parameter [slot number] is used in case 2 the same tools are mounted on the machine and a specific tool has to be used. For example when 2 pens are mounted and the one on slot position 1 needs to be used, then following command can be used: SP19,1;

Slot numbers: 1 = left slot : 2 = Middle slot: 3 = right slot.

Tool Number	Tool Names
0	Pen
1	Drag Knife
2..9	<i>Reserved</i>
10	Any cut tool detected on the machine
11..13	<i>Reserved</i>
14	Camera
15	Pointer
16..18	<i>Reserved</i>
19	Cut Out Knife
20	Kiss Cutting Knife
21	Creasing Wheel
22	Oscillating Knife
23..29	<i>Reserved</i>
30	Creasing Wheel type 2
31	Creasing Wheel type 3
32	V-Cut 45
33	V-cut 30
34	V-Cut 22.5
35	V-Cut 15
36	V-cut 0
37	Router

Table: Tool numbers and tool names

2.1.8 SQ tool; Tool check Command

The *SQtool;* command checks if a tool is available on the machine. If it is not available the machine will give a warning and cancel the job. It does not select a tool. It is strongly advised to send the appropriate tool check command for each tool used in the current job before outputting the job. [See table tools numbers and names](#) for the list of tool numbers.

For example if a job needs a pen and a cut out tool, at the beginning of the job send *SQ0; SQ19;* The machine will then check if the tools are fitted on the machine before starting the job. This command is optional.

2.1.9 VS n; Velocity Command

To set the cutting velocity. n specifies the velocity of the device in cm/s.

2.1.10 RP n; router rotation speed Command

To set the speed of the rotation of the router. This command is ignored when a standard router is used, only the HF router. n specifies the velocity of the device in rpm.

2.1.11 BC; Bit change command Command

To pause the job and give a signal that the user can change the routing bit.

2.1.12 ZP n,m; Z axis position Command

To set the cutting depth. n value is neglected, m specifies offset from the cutting depth. The table only accepts negative m values (cutting less deep), if a positive value is used, then the firmware in the table will automatically 'add' a minus before the value. change in cutting depth applies to the current tool.

2.1.13 OH; Output Hardclip Command

The *OH;* output hardclip command returns the size of the loaded media.
Data returned by the cutter:

Window Lower Left X	Window Lower Left Y	Window Upper Right X	Window Upper Right Y	Carriage Return
---------------------------	---------------------------	----------------------------	----------------------------	--------------------

Table - OH Report Format

2.1.14 OA; Output Actual Position

Returns the actual position and up-down status of the tool. All instructions prior to this OA will first be completed before the machine responds. Data returned by the cutter:

x,y,p\r\n x and y are the plotter coordinates and p is 1 if tool status is down, 0 if tool status is up.

2.1.15 FF; Feed Command and FLx; command

If a pneumatic pack and/or conveyor are installed on the cutter, then media can be fed through the machine. Feeding is started by the FF; command. The Feeding length is controlled by the FLx; command.

In the FLx; command, x is the length to feed. A negative feed Length is allowed and will result in a negative feed length, a Feed Length bigger than the table length is allowed, the feeding will then take place is several steps.

2.1.16 PB Vacuum control

The vacuum of the machines turns automatically on, as soon as the machine moves. The vacuum will automatically turn off after 100 seconds of inactivity. It is however possible to turn the vacuum On or off using the PB command.

PB 2,0; Turns the vacuum off.

PB 2,1; turns the vacuum on.

2.1.17 PS Vacuum direction.

The PS command controls the direction of the vacuum.

PS 1,1; will make sure the vacuum is set in suction position.

PS 1,0; will turn the vacuum in blowing mode. This to allow to remove material from the bed more easily.

Note: When cutting the vacuum is automatically set in suction mode.

2.1.18 ZT0; Park heads

ZT 0; Parks the heads in up position. It is best to end the job with this command.

2.1.19 AUbeta; define lifting angle

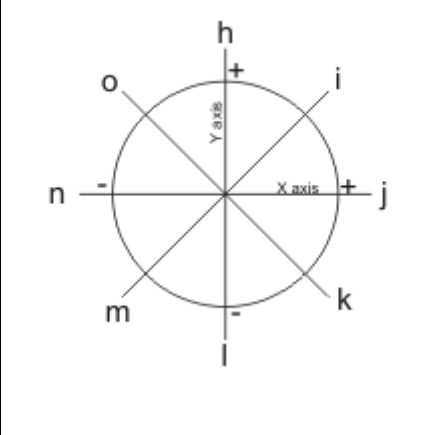
This command defines the maximum angle between 2 vectors. If the angle between 2 vectors is bigger than this angle, then at the end of the first vector, the tool is raised, rotated in the correct direction and lowered to continue cutting.

2.1.20 OVn; Overcut distance

Sets the overcut distance (unit = mm).

2.1.21 LMn; Local move commands

When a local move command is sent to the flatbed cutter, the cutter will start moving its cutting head, in the direction of the specified command, until the cutter receives the end character "#".

	DM/PL command	Cutter movement
	h	+Y
	i	+X,+Y
	j	+X
	k	+X,-Y
	l	-Y
	m	-X,-Y
	n	-X
	o	-X,+Y

2.1.22 MStext MESSAGE

Sends a message (text string) to the user interface, the plotter will continue when the message has been acknowledge on the user interface.

Example MStext material;

2.1.23 Sample HP-GL File

IN; PA;PU1000,1000;PD2000,2000;PD 2000,0 ZT0;

In this example, the IN; resets the device. Absolute knife positioning is selected with the PA; command. The knife is sent in up position to absolute co-ordinate 1000,1000. The knife is lowered with the Down PD command and a line is cut to absolute co-ordinate 2000,2000 then a line is cut to position 2000,0.

2.2 Encapsulated Language

The cutter has a specific language to control some settings of the cutter, this can be used to set settings such as speed, we however advise to use the HPGL commands for this.

2.3 Contour Cutting

Contour cutting is done using printed registration marks. These marks are then recognized by the camera installed on the machine. SummaFlex Pro can be used in order to read the markers. This program does not come for free. It is also possible to write own routines for recognition of the marks.

2.3.1 Mark recognition using Custom software.

The camera is not connected to the controller of the cutter, it is directly connected to the computer via a USB 2.0 port. USB drivers are only available for Microsoft Windows operating systems (up to Windows 10). There are no drivers available yet for Mac OS or Linux operating system.

It is possible to read the video stream directly into your software. This is done using the **DirectShow** feature of the Windows operating software. Commercial libraries are also available to speed up this process.

You need to do the following in order to do contour cutting using the camera.

Read the markers using DirectShow and by moving the cutting head to the next markers (using DMPL or HPGL commands).

Transform the cutting data prior to sending it to the cutter, correction for rotation, origin, scaling will be necessary.

Send the transformed data to the cutter.

All of this this is quite some work, but it allows the use of custom markers, it allows more flexibility in your software, it allows better integration within your software.

2.3.2 Mark recognition using SummaFlex Pro.

SummaFlex Pro support the mark recognition and also transforms the cutting data. It is advised to use the OXF format, SummaFlex however also supports other formats, including plt, pdf and eps.

The markers can be any shape, For SummaFlex Pro however it is strongly recommended to have **circles of 5mm** in a color which has a big contrast with the underlying material on which it is printed. The marks can be placed anywhere in the design, at least 3 are needed and should include the whole design for better accuracy.

2.4 USB

2.4.1 Introduction.

The usb communication on the Flatbed cutters is the same as on the other cutters from summa, the summacut and S class. So the same routines can be used.

First some basics about the USB bus and the support of windows for this bus are described.

Then the specific details of the software-architecture for communication with the cutters are described.

The document ends with a sample written in C that clarifies the theory.

2.4.2 USB and Windows

USB is a new standard bus available on all new PC's. This bus offers serial communication at high speed (12 MBit/s). It is hot pluggable, which means devices can be attached or removed at any time from the PC.

Only the newest versions of Windows (Windows 98 and Windows 2000) fully support the USB bus. The USB drivers are only 32-bit compatible, this means that no 16-bit program can communicate over the USB bus.

The Win32 API (or let say Windows) offers some 32 bit functions to communicate. These functions are CreateFile(), ReadFile(), WriteFile() and CloseHandle().

More info about these functions can be found in your development tools (Visual C++ or Borland C++ or MSDN). The term 'file' also includes communications resources such as the serial RS232 port (COM port), the parallel port (LPT1) and the USB port.

The CreateFile() is used to get a handle to a file or communication resource. The WriteFile() is then used to send data to the opened 'file'. The CreateFile() function can **NOT** be used directly with the USB bus. Instead a function (open_file()) provided by Summa must be used. This function does the same as CreateFile(); it returns a handle to the USB bus for the cutter. Then the WriteFile() and ReadFile() functions can be used.

2.4.3 USB Pipes

Each USB device (in our case the cutter) communicates to the host software (= software on PC) through what is called pipes. Most USB devices do have several pipes implemented. The cutter has 3 pipes implemented. This means that there are 3 communications channels between the cutter and the host software.

The first one is called the Default Control Pipe, the operating software (Windows) uses this pipe for plug and play. This pipe is used to get information about the USB configuration of the cutter. Third party developers do not need this pipe.

The two most important pipes are PIPE00 and PIPE01. PIPE01 is an **unidirectional** pipe that transports data from the host PC to the cutter. This data is the cut data (DM/PL or HP-GL). PIPE00 is also **unidirectional** and carries information from the cutter to the host. For example you can get the size of the media through this pipe (in DM/PL for example).

So changes for the software in comparison to a RS232 serial port will be that sending data has to be handled over PIPE01 and receiving data will be handled over PIPE00. Each pipe will need a different handle compared to RS232.

As an example to get the paper size of the cutter (in DM/PL) you will have to send ';;ER' on PIPE01 and you will have to read the response on PIPE00.

The third pipe has been called PIPE02. Reading from this pipe returns the free space in the internal buffer of the cutter. This pipe can be used to implement a software handshake method. This can be useful as the WriteFile() function only returns when the data that has to be sent is completely transmitted.

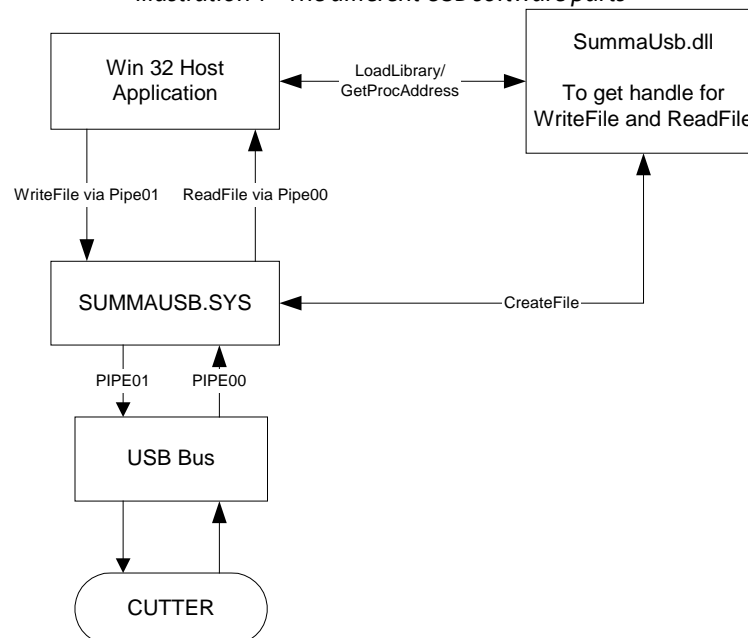
2.4.4 Win32 Software Components

Summa has written a windows driver (SummaUsb.sys) that allows Win32 programs to communicate with the cutter over the USB bus with the standard Win32 API functions ReadFile(), WriteFile() and CloseHandle(). Before using this functions you must get a handle to the USB cutter device. For this use 'open_file()', a function that Summa has written. This function is made available through the SummaUsb.dll. (see the sample on how to use the DLL).

SummaUsb.dll is a Win32 dynamic link library and calls specific functions that only Win98 and Windows 2000 does support, so you will get an error when trying to load the SummaUsb.dll with LoadLibrary() when your software is running in Win95 or WinNT 4.0. So before loading this DLL you should check the windows version. If it is WinNT 4.0 do not load the library or the user will get an error message on his screen! Windows 95 does not give an error message.

16-bit programs are not supported. You must use the Win32 WriteFile() and ReadFile() functions to communicate over the USB bus.

Illustration 1 - The different USB software parts



2.4.5 SummaUsb.sys

The Plug and play Manager from windows automatically load this WDM driver when it detects that a cutter has been plugged into the USB bus. This driver will be shipped with every cutter on a Companion CD.

NOTE : *The USB bus allows hot plugging and unplugging of the devices attached to its bus. The cutter may be plugged in after your program has started. You will not get a handle when the cutter is not plugged in or is powered off. You will only get a handle to the device after the cutter is powered on and plugged in the USB bus.*

In comparison you do always get a handle to a RS232 COM serial port. This is an important difference that your software should handle. This can best be done by getting a handle to the USB device just before sending/receiving data. It is preferred not to get a handle at initialization of your program.

2.4.6 SummaUsb.dll

SummaUsb.dll is a dynamic link library that helps you to get an easy access to the handles of the USB pipes from the cutter. You can get 2 handles to the cutter, one for each pipe. After you have a handle to the USB cutter, you don't need SummaUsb.dll anymore. You can use the standard win32 API functions ReadFile() and WriteFile() to communicate with the cutter.

You will have to include this file in the installation of your product.

This DLL has 1 functions called open_file(), that does the same as CreateFile():

HANDLE __stdcall open_file (**char*** lpFileName, **DWORD** dwFlagsAndAttributes);

Parameters

lpFileName

Points to a string that specifies the name of the pipe. The only valid values for that parameter are the strings "PIPE00" and "PIPE01". "PIPE00" to read data from the cutter and "PIPE01" to write data to the cutter.

dwFlagsAndAttributes

Instructs the system whether or not to use asynchronous communication. See information about CreateFile().

Values: 0(NULL) for synchronous operation or FILE_FLAG_OVERLAPPED for asynchronous operation.

Return Values

If the function succeeds, the return value is an open handle to the specified pipe. If the function fails, the return value is INVALID_HANDLE_VALUE. To get extended error information, call GetLastError(). The pipe cannot be opened when the cutter is powered off or when the cutter is not connected to the USB bus.

Note : *This function must be declared as __stdcall.*

WriteFile() and ReadFile() can then use the handle returned by open_file().

2.4.7 Win32 Host Application

There are some restrictions on the Win32 API functions `ReadFile()` and `WriteFile()`.

- **Writing Data To Cutter.**

You must send the data in little packets (e.g. 256 bytes). This gives much better performance. When sending a file of 1 Megabyte, for instance, the USB driver stack will take up too much time and your computer will seem to hang. When sending little chunks it is also easier to cancel a current job being sent.

When the data is split in packets of 256 bytes, the last packet will probably be smaller than 256 bytes, this is no problem.

If the input buffer of the cutter is full, the `WriteFile()` function will not return, but it will wait until some place in the buffer is available.

- **Reading Data From Cutter**

The timeout principle doesn't exist on the USB bus, at least not in the same way as in the serial port. So when querying info from the cutter, you normally first send data on `PIPE01` and then read data on `PIPE00`. The cutter needs some time to put data on the USB bus after receiving the request. While the data is not ready the cutter will respond to any query with a zero length packet. This means the `ReadFile()` function will return with 0 bytes read. You will have to call `ReadFile()` until you get the desired response from the cutter. Calling `ReadFile()` only once will not be enough, the cutter needs in most cases to have 2 `ReadFile()` commands, the first one will respond with 0 bytes, the second one with the desired response (if the cutter is ready to answer).

So when getting an answer from the cutter of zero bytes or less than expected, does not mean that no data is available anymore. The cutter needs some time to put the data in its output buffer. This problem is handled in the sample by adding some delays (see function `delaysms()`).

Reading data must be done by multiples of 16. Do not read 1 byte at a time with the `ReadFile()` function! When reading more data than available, the `ReadFile()` will return with success, but the number of bytes read will be set to what is really read.

2.4.8 Plug And Play Considerations

USB claims to be plug and play, this means that the user just should plug in the cutter and that all drivers are installed automatically. This is the case for the low level USB driver (`SummaUsb.sys`). But the user still needs to install the correct vector driver within your program (e.g. a DM/PL or HP-GL driver for Summa), the user also needs to select to which port the cutter/printer is installed.

When using USB you can also introduce plug and play for your software. This means that your software can automatically choose the correct driver when a cutter from Summa is attached on the USB port without the intervention of the user. To do this you need to check if a cutter from Summa is attached on the USB port. This will be the case when you get a handle to the USB port. When getting this handle you know for sure that a cutter from Summa is attached to the USB port. You could check this for example every time your program is started. Once the cutter is found, you can then install the correct vector driver and select the USB port. The installation of the driver should only occur once of course and not every time your program is started.

2.4.9 Summary

- Make a 32-bit program (Win32) using ReadFile() and WriteFile() to communicate.
- Load the DLL SummaUsb.dll using LoadLibrary().
- Get the address of the function open_file() using GetProcAddress().
- Use open_file() from SummaUsb.dll to get a handle to the USB port instead of CreateFile().
- Open 2 handles, one for reading data and one for writing data.
- Send data in little chunks (e.g. 256 bytes) with WriteFile().
- Read data in multiples of 16 bytes with ReadFile().
- When reading data the cutter may respond with 0 bytes, read more than once, before deciding the cutter doesn't respond.
- USB has no time-out function (like the serial port)

2.4.10 Win32 Sample Application

The sample is compiled using Visual C++ 5.0. The sample is a win 32 console application (runs in a 'DOS' box). Files are distributed together with this document. The RWSumma.c file gives an example on how to open handles to the USB pipes and how to send and read data. The sample also shows how to use SummaUsb.dll and its function "open_file()".

The method used in this sample is the so-called Run-Time Dynamic Linking. It has the advantage that the process can continue running even if the DLL is not available. The program can then notify the user of an error. If the user can provide the full path of the missing DLL, the process can use this information to load the DLL even though it is not in the normal search path.

This sample does not allow to cancel data or stop the execution of the ReadFile() or WriteFile() function. It has no time-out function implemented.

```
/*++
Copyright (c) 1999 Summa N.V.
Module Name:
    RWSumma.c
Abstract:
    Console test app for SummaUsb.sys driver
Environment:
    user mode only
Notes:
    Copyright (c) 1999 Summa N.V. All Rights Reserved.
```

```
Revision History:
    11/11/99: created
--*/
```

```
#include <windows.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <time.h>
#include <sys\timeb.h>
#include <basetyps.h>
```

```
char inPipe[32] = "PIPE00";    // pipe name for bulk input pipe on
our test board
char outPipe[32] = "PIPE01";   // pipe name for bulk output pipe
```

```
on our test board
int gDebugLevel = 1;      // higher == more verbose, default is 1, 0
                             turns off all
int WriteLen = 0;         // #bytes to write
int ReadLen = 0;          // #bytes to read

// prototype for open_file function from summausb.dll
typedef HANDLE (__stdcall *OPENFILE)(char*, DWORD);
OPENFILE POpenFile;

// functions
/*++
Routine Description:
    waits for a time ms (in milliseconds)
Arguments:
    ms : time to wait in milliseconds
Return Value:
    Zero
--*/
void delaysms(double ms)
{
    double elapsed_time = 0;
    struct _timeb start, finish;

    _ftime( &start );
    _ftime( &finish );

    do
    {
        _ftime( &finish );
        elapsed_time = ((finish.time - start.time) * 1000) +
finish.millitm
start.millitm;
    } while (elapsed_time < ms);
}

int _cdecl main(
    int argc,
    char *argv[])
/*++
Routine Description:
    Entry point to RWsumma.exe
    Sends data to the USB cutter and reads response.
Arguments:
    argc, argv standard console 'c' app arguments
Return Value:
    Zero
--*/
{
    char *pinBuf = NULL, *poutBuf = NULL;
    int nBytesRead, nBytesWrite, TotalBytesRead;
    ULONG i;
    UINT success;
```

```
HANDLE hRead = INVALID_HANDLE_VALUE, hWrite = INVALID_HANDLE_VALUE;
ULONG totalBytes = 0L;
char DebugString2[] = " \x1B;@:.MENU. ";
char DebugString[] = ";; EC1 ER ";

UINT bResult = 0;

HINSTANCE hinstLib;
BOOL fRunTimeLinkSuccess = FALSE;

// First the SummaUsb Dll will be loaded
// then we will get the adress of the function needed in the dll

// Get a handle to the DLL module.
hinstLib = LoadLibrary("summausb.dll");

// If the handle is valid, try to get the function address.
// Note:in Win95 or Win Nt 4.0, the handle will be incorrect.
//      in 16 bit application the handle will also be incorrect
//      (hinstLib will be smaller than 32)
if (hinstLib != NULL)
{
    POpenFile = (OPENFILE) GetProcAddress(hinstLib, "open_file");

    // If the function address is invalid, print a error message

    fRunTimeLinkSuccess = (POpenFile != NULL);
    // If unable to call the DLL function, DLL must be corrupted?
    if (! fRunTimeLinkSuccess)
        printf("Error : could not open function OpenFile");
}
else
{
    printf("Error : could not open Summausb.dll\n");
    return (1);
}

//
// open the Read and Write Pipes
//

hWrite = (POpenFile)( outPipe, (DWORD)NULL);

if (hWrite == INVALID_HANDLE_VALUE)
{
    printf("could not open %s", outPipe);
    return 0;
}

hRead = (POpenFile)(inPipe, (DWORD)NULL);

if (hRead == INVALID_HANDLE_VALUE)
{
    printf("could not open %s", inPipe);
```

```
        return 0;
    }

    //
    // put some data in the output buffer
    //

    WriteLen = sizeof(DebugString);
    poutBuf = malloc(WriteLen);
    sprintf(poutBuf, DebugString);

    // allocates some memory to put read data.
    ReadLen = 64;
    pinBuf = malloc(ReadLen + 1);

    if ( poutBuf && hWrite != INVALID_HANDLE_VALUE)
    {
        // skip any data that is still left in the buffer of the cutter
        // by reading data from the cutter, until no more data is
        available

        do
        {
            success = ReadFile(hRead, pinBuf, ReadLen, &nBytesRead,
NULL);
            delaysms(10);
            /* add some kind of timeout or abort procedure ? */
        } while (nBytesRead);

        //
        // send data to the cutter
        //
        WriteFile(hWrite, poutBuf, WriteLen, &nBytesWrite, NULL);
        printf("<%s> W (%04.4d) : request %06.6d bytes -- %06.6d
byte\n",
                                outPipe, i, WriteLen,
nBytesWrite);
    }

    if (pinBuf)
    {
        nBytesRead =0;

        /* read untill we get some answer from the cutter */
        /* the cutter will return nBytesRead = 0 as long as no data is
        available*/

        while (!nBytesRead)
        {
            success = ReadFile(hRead, pinBuf, ReadLen, &nBytesRead,
NULL);
            /* add some kind of timeout or abort procedure ? */
        }
    }
}
```

```
        // we have some data, process it
        pinBuf[nBytesRead] = 0;
        printf(pinBuf);
        printf("\n\r");
        delays(20);    // add some delay to allow the cutter to
prepare its next data

        TotalBytesRead = nBytesRead;

        /*if more data needed read, read more data,
        The cutter will return 0 bytes, while it is preparing its
data !!
        */

        while (nBytesRead)
        {
            success = ReadFile(hRead, pinBuf, ReadLen, &nBytesRead,
NULL);
            TotalBytesRead += nBytesRead;
            pinBuf[nBytesRead]=0;
            printf(pinBuf);
            delays(20);
        }

        printf("\n<%s> R (%04.4d) : %06.6d bytes read\n", inPipe, i,
TotalBytesRead);
    }

    if (pinBuf)
    {
        free(pinBuf);
    }

    if (poutBuf)
    {
        free (poutBuf);
    }

    // close devices if needed
    if (hRead != INVALID_HANDLE_VALUE)
        CloseHandle(hRead);
    if (hWrite != INVALID_HANDLE_VALUE)
        CloseHandle(hWrite);

    // free the dll Library
    if (hinstLib)
        FreeLibrary(hinstLib);

    return 0;
}
```

2.4.11 Handshake Sample

The following application shows how to implement a handshaking method. It also uses the Run-Time Dynamic Linking method.

The program starts with creating a handle `fp` to the file specified in the command-line argument `argv[1]`. It then loads the library `SummaUSB.dll` and stores the handle in `hLib`. This handle is then used to retrieve the address of the function `open_file` to get a handle to an USB-pipe. This address `pOpenFile` is then used to get a handle to the USB output-pipe `hWrite` and status-pipe `hStatus`.

Now we are ready to transmit the data to the cutter.

First, we check if we're not at the end of a file, then we read a chunk of data (up to 256 bytes). If we could get the data, we have to check if there is space for it in the cutter's buffer. This is accomplished by reading from the USB status-pipe `hStatus` with the `ReadFile()` function. The data `szBuffer` returned represents the free space in the cutter's internal buffer. The converted value `nBufferFree` is compared against the amount of data `iCount` we've previously read. When there's not enough space for the data, we enter a loop until space becomes available. As soon as there's enough space for the data, we can send it to the cutter using the `WriteFile()` function with a handle to the USB output-pipe `hWrite`.

All this is repeated until all data has been sent to the cutter.

Next, all handles are closed and the Summa USB library `hLib` is unloaded.

```
// File2USB.cpp : Defines the entry point for the console
application.

#include <windows.h>
#include <stdlib.h>
#include <stdio.h>

char    inPipe[32] = "PIPE00" ; // pipe to read responses from the
cutter
char    outPipe[32] = "PIPE01" ; // pipe to send data to the cutter
char    statusPipe[32] = "PIPE02" ; // pipe to query free space in the
cutter's internal buffer

// prototype for open_file function from summausb.dll
typedef HANDLE (__stdcall * OPENFILE)(char *, DWORD) ;
OPENFILE pOpenFile ; // pointer to the open_file function

int main(int argc, char* argv[])
{
    FILE *fp ;
    HMODULE hLib ; // The SummaUSB library handle
    HANDLE hWrite ; // A handle to the USB output pipe
    HANDLE hStatus ; // A handle to the USB status pipe
    char szData[256] ; // The data buffer
    unsigned int iCount ; // Amount of data we've read
    char szBuffer[16] ;
    unsigned long n ;
    unsigned long nBufferFree ; // Free space in the cutter's internal
buffer
    BOOL bSuccess ;
```

```
// A filename is required
if (argc != 2)
{
    printf("Usage : File2USB <filename>\n");
    return 1 ;
}

// Open the specified file
fp = fopen(argv[1], "rb") ;

if (fp == NULL)
{
    printf("Error: could not open file \"%s\"\n", argv[1]) ;
    return 2 ;
}

// Load the library which connects our app to the USB driver
hLib = LoadLibrary("SummaUSB.dll") ;

if (hLib == NULL)
{
    fclose(fp) ;
    printf("Error: could not open SummaUSB.dll\n") ;
    return 3 ;
}

// Get the address of the function which gives us a handle to the
specified pipe
pOpenFile = (OPENFILE)GetProcAddress(hLib, "open_file") ;

if (pOpenFile == NULL)
{
    fclose(fp) ;
    printf("Error: could not retrieve address of open_file
function\n", outPipe) ;
    return 4 ;
}

// Get a handle to the pipe where we can put our data
hWrite = (pOpenFile)(outPipe, 0) ;

if (hWrite == INVALID_HANDLE_VALUE)
{
    fclose(fp) ;
    printf("Error: could not open pipe \"%s\"\n", outPipe) ;
    return 5 ;
}

// Get a handle to the pipe where we can read the free space in
the cutter's internal buffer
hStatus = (pOpenFile)(statusPipe, 0) ;

if (hStatus == INVALID_HANDLE_VALUE)
```



```
{
    fclose(fp) ;
    printf("Error: could not open pipe \"%s\"\n", statusPipe) ;
    return 6 ;
}

bSuccess = TRUE ;

// Repeat until no more data available or when there are problems
on the USB port
while (!feof(fp) && bSuccess)
{
    // Get some data to send
    iCount = fread(szData, sizeof(char), sizeof(szData), fp) ;

    // If there is something to send
    if (iCount > 0)
    {
        // Use software handshaking
        do
        {
            // Read data from USB status pipe
            bSuccess = ReadFile(hStatus, szBuffer,
sizeof(szBuffer), &n, 0) ;
            szBuffer[n] = 0 ; // Terminate string
            nBufferFree = atol(szBuffer) ; // Free space in
internal buffer of the cutter

        } while (bSuccess && nBufferFree < iCount) ; // Repeat
until space becomes available

        // Send the data through the USB output pipe
        bSuccess = WriteFile(hWrite, szData, iCount, &n, 0) ;
    }
} ;

CloseHandle(hStatus) ;
CloseHandle(hWrite) ;
fclose(fp) ;

// Release the SummaUSB library
FreeLibrary(hLib) ;
return 0 ;
}
```

2.5 Wii Remote

Axis Control uses a Wii Remote as an extension of the keyboard.
This can be used for example to set the origin of the machine.

All Wii Remote keys are hooked to virtual key codes:

A	VK_ENTER
B	VK_SHIFT
1	VK_F1
2	VK_F2
-	VK_PRIOR
+	VK_NEXT
Home	VK_ESCAPE
Up arrow	VK_UP
Down arrow	VK_DOWN
Left arrow	VK_LEFT
Right arrow	VK_RIGHT

Other programs can catch and process the Wii Remote key events.

1. Register our event "AxisControlCatchWiiRemote" to get the right message ID.

```
UINT Msg = RegisterWindowsMessage("AxisControlCatchWiiRemote");
```

2. To start catching Wii Remote key events.

```
PostMessage(HWND_BROADCAST, Msg, 1, (long)(...your window handle...));
```

3. To stop catching Wii Remote key events.

```
PostMessage(HWND_BROADCAST, Msg, 0, 0);
```

Do not forget to stop catching Wii Remote key events, else Axis Control will not work properly anymore until restarted!

3 The OXF format

This format is supported by SummaFlex and SummaFlex Pro. It is the advised format to use when working with SummaFlex.

The data format: HPGL, with a few extras.

File extension: .OXF (OptiSCOUT exchange file)

Resolution: 0.01 mm.

Different layers are separated by HPGL "SP" commands. The layer 99 (SP99) must contain the registration marks. These marks must be circles of 5 mm.

The OXF format is explained in another document.